Vers des architectures acteur-critique neuronales efficaces en données

Matthieu Zimmer, Yann Boniface, Alain Dutech

Vers des architectures acteur-critique neuronales efficaces en données ⋆

Matthieu Zimmer[1], Yann Boniface[1], et Alain Dutech[2]

[1] Université de Lorraine, LORIA, UMR 7503, F-54000, Nancy, France

[2] INRIA, LORIA, UMR 7503, F-54000, Nancy, France
{prenom}.{nom}@loria.fr

Résumé : Un nouvel algorithme d'apprentissage par renforcement, traitant à la fois des espaces continus d'états et d'actions, est proposé. Il ne nécessite pas de connaître *a priori* des états buts ou de bonnes trajectoires pour fonctionner. Les besoins en connaissances expertes sur le domaine appliqué sont minimales grâce à l'emploi d'estimateur non-linéaire : des réseaux de neurones. Ce nouvel algorithme est *on-policy*, *hors-ligne*, *sans modèle* de l'environnement. Il produit des politiques stationnaires en temps discret et déterministes en maximisant la somme actualisée des récompenses. Des résultats expérimentaux montrant la bonne performance de l'algorithme sont présentés sur deux environnements déterministes : acrobot et cartpole.

Mots-clés : Apprentissage par renforcement, Acteur-Critique, Espaces continus.

1 Introduction

L'Apprentissage par Renforcement (APR) (Sutton & Barto, 1998) est un cadre pour la résolution de problèmes à décision séquentielle où un agent interagit avec son environnement et adapte sa politique en fonction d'un signal de récompense scalaire. Les agents apprenant par renforcement sont capables d'apprendre de manière autonome des tâches difficiles, comme naviguer dans un labyrinthe ou jouer à des jeux vidéos (Mnih *et al.*, 2015). Alors que les bases de l'APR sont maintenant bien établies, un certain nombre de chercheurs a étudié des variantes où les environnements avec des espaces continus conduisent à des problèmes de plus en plus pratiques.

Ainsi, le but de cet article est de présenter un algorithme d'APR en respectant deux exigences principales : 1) traiter des espaces continus d'états et d'actions afin de résoudre des problèmes plus réalistes, 2) la connaissance ajoutée par le concepteur à l'agent devrait être minime pour permettre à l'agent d'acquérir progressivement ses propres représentations dans une perspective de robotique développementale (Asada *et al.*, 2009).

Tout d'abord, le cadre des algorithmes classiques d'APR et leurs limites sont décrites. Ensuite, l'algorithme principal de cet article est exposé avec une comparaison expérimentale sur de multiples environnements.

2 Cadre formel

2.1 Apprentissage par renforcement

L'APR est un cadre formel qui modélise les problèmes de décisions séquentiels, dans lequel un agent apprend à prendre de meilleures décisions en interagissant avec l'environnement. Lorsque l'agent effectue une action, l'état change et l'agent reçoit une valeur scalaire, pouvant être nulle, appelée récompense, encodant les informations sur la qualité de la transition. L'objectif de l'agent est de maximiser sa récompense totale à long terme.

⋆. Les données ont été analysées numériquement à l'aide du logiciel libre GNU Octave (John W. Eaton David Bateman & Wehbring, 2015). Les expériences présentées dans ce document ont été effectuées en utilisant le banc d'essai Grid'5000, soutenu par un groupe d'intérêt scientifique organisé par l'Inria, le CNRS, RENATER et plusieurs universités ainsi que d'autres organisations (voir https://www.grid5000.fr).

Le formalisme sous-jacent de l'APR est celui des Processus Décisionnels de Markov (PDM). Un PDM est formellement défini comme étant un n-uplet $\langle S, A, T, R\rangle$ où S est un ensemble d'état, A un ensemble d'action, $T : S \times A \times S \to [0, 1]$ sont les probabilités de transition entre états ($T(s, a, s') = p(s'|a, s)$ est la probabilité d'atteindre l'état s' à partir de l'état s après exécution de l'action a) et $R : S \times A \to \mathbb{R}$ est un signal de récompense. Une *politique* $\pi : S \times A \to [0, 1]$ code la façon dont l'agent va se comporter (la probabilité de prendre une action dans un état donné). Une politique *optimale* π^* maximise la récompense actualisée prévue, à savoir :

$$\pi^* = \arg\max_{\pi} J(\pi) = \arg\max_{\pi} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \times R(s_t, \pi_t(s_t))\Big] \tag{1}$$

où t représente un pas de temps et $0 < \gamma < 1$ est un facteur d'actualisation. Dans le cadre de l'APR, le but est d'apprendre une politique optimale lorsque le modèle, T et R, est inconnu.

2.2 Traitement des espaces d'état continus

Lorsque l'espace S est continu, les méthodes d'APR classiques comme Least-Squares Temporal Difference (LSTD) (Bradtke *et al.*, 1996) reposent sur une estimation des fonctions de valeur : $Q : S \times A \to \mathbb{R}$, les valeurs séquentielles des actions dans chaque état, ou $V : S \to \mathbb{R}$ les valeurs de chaque état.

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s, a_t = a\Big] \tag{2}$$

Cette estimation est habituellement une combinaison linéaire de fonctions de base fournies, qui doivent être soigneusement définies par un expert. Leur nature et leur nombre limitent l'expressivité de la fonction apprise. Cela entrave la capacité de l'agent à développer ses propres représentations. Afin de minimiser les connaissances apportées par les experts (exigence 2), l'emploi de fonctions non-linéaires (tel que les réseaux de neurones) sera explorée car elles ont une plus grande gamme d'expressivité et évitent la définition *a priori* des fonctions de base.

2.3 Traitement des espaces d'action continus

2.3.1 Critique-seul

Les méthodes à *critique-seul* dérivent une politique directement à partir de la fonction de valeur :

$$\pi(s) = M(Q, s, A) = \arg\max_{a \in A} Q(s, a) \tag{3}$$

où la fonction M peut être l'opérateur $\arg\max$, Boltzmann soft max, etc. Plusieurs algorithmes (Sutton & Barto, 1998) ont été conçus avec différentes règles de mise à jour : Q-Learning, Sarsa, LSTD, etc. Parmi eux, Fitted Q Iteration (FQI) (Riedmiller, 2005) est particulièrement intéressant car il utilise des réseaux neuronaux non-linéaires, est efficace en données et tire parti de l'algorithme de backpropagation résiliant (RPROP) (Igel & Hüsken, 2000; Riedmiller & Braun, 1992).

Le principal problème de ces méthodes est qu'elles sont très difficiles à appliquer à des espaces *d'action continus*. L'opérateur $\arg\max$ ne peut pas être utilisé et l'optimisation de la fonction Q, une combinaison linéaire de fonction de base non linéaires, est une tâche complexe.

2.3.2 Acteur-seul

De l'autre côté, il est possible de définir une politique paramétrée π_θ sans fonction de valeur (Sutton *et al.*, 1999). L'objectif est alors de trouver les meilleurs paramètres θ d'après la fonction de coût $J(\pi_\theta)$ (1) en explorant l'espace des paramètres de dimension finie Θ. Les algorithmes évolutionnaires tel que Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen & Ostermeier, 2001) ou de descente de gradient sont des outils communs d'optimisation. Les principaux inconvénients des méthodes acteur-seul sont la grande variabilité du coût J, et pour les méthodes basées sur les gradients, l'effet de plateau et des minima locaux qui peuvent conduire à des politiques peu performantes (Grondman *et al.*, 2012; Konda & Tsitsiklis, 1999).

2.3.3 Acteur-Critique

Les architectures acteur-critique (Konda & Tsitsiklis, 1999) tentent de combiner les avantages des deux méthodes précédentes. Le critique apprend la fonction de valeur pour réduire la variabilité de l'approximation de J et l'acteur apprend la politique permettant l'utilisation d'actions continues. Continuous Actor Critic Learning Automaton (CACLA) est un algorithme acteur-critique (Van Hasselt & Wiering, 2007) utilisant des réseaux de neurones à la fois pour le critique et pour l'acteur. Cependant, c'est un algorithme *en ligne* qui n'est pas efficace en données puisque chaque interaction produite n'est utilisée qu'une seule fois.

3 Neural Fitted Actor-Critic

Neural Fitted Actor-Critic (NFAC) est un nouvel algorithme acteur-critique s'inspirant à la fois de CACLA et FQI. Il est *hors ligne* et peut réutiliser les données et traiter les espaces continus. Il procède en deux étapes, résumées dans la Figure 1 : 1) étant donné la politique courante π, il échantillonne et recueille des données d'interactions dans $\mathcal{D}_\pi$, 2) améliore l'approximation du critique et la politique de l'acteur à l'aide de $\mathcal{D}_\pi$.

Collecte des données d'interaction D_π est composé de plusieurs n-uplet $(s_t, u_t, a_t, r_{t+1}, s_{t+1})$. Pour un état s_t, l'acteur fournit la meilleure action courante u_t, qui est légèrement modifiée en une action exploratoire a_t (pouvant être égale à u_t), qui conduit à un nouvel état s_{t+1} avec récompense r_{t+1}.

Estimation des réseaux neuronaux L'acteur et le critique sont mis en oeuvre en utilisant des perceptrons multicouches. Leur mise à jour, qui est détaillée ci-dessous, utilise la backpropagation RPROP (Igel & Hüsken, 2000; Riedmiller & Braun, 1992) évitant la définition des taux d'apprentissage grâce à l'apprentissage hors-ligne.

3.1 Mise à jour du critique

La mise à jour du critique repose sur plusieurs itérations d'apprentissage supervisé comme cela se fait dans FQI. À chaque itération k, une base d'apprentissage $\{(s_t, v_{k,t})\}$ de $(S \times \mathbb{R})^{|\mathcal{D}_\pi|}$ est construite à partir de $\mathcal{D}_\pi$ en utilisant l'approximation courante de V_k. Chaque état s_t de $\mathcal{D}_\pi$ est associé à une cible $v_{k,t}$ qui est soit $r_{t+1} + \gamma V_k(s_{t+1})$ soit r_{t+1} lorsque s_{t+1} est un état goal ou un état absorbant. Ainsi V_{k+1} est calculé comme suit :

$$V_{k+1} = \underset{V \in \mathcal{F}_c}{\operatorname{argmin}} \sum_{s_t \in \mathcal{D}_\pi} \Big[V(s_t) - v_{k,t} \Big]^2 \tag{4}$$

$\mathcal{F}_c$ est l'espace de recherche du critique (par exemple un perceptron multicouche). La base d'apprentissage doit être reconstruite à chaque itération puisque les cibles $\{v_{k,t}\}$ pour apprendre V_{k+1} dépendent de V_k. Contrairement à FQI qui est *off-policy* et en raison de l'architecture acteur-critique, V est ici une évaluation *on-policy* de l'acteur courant. Cela implique que D_π doit être reconstruit et nettoyé après chaque modification de l'acteur (à la fin de chaque épisode).

3.2 Mise à jour de l'acteur

La mise à jour de l'acteur modifie la politique afin de renforcer les actions exploratoires a_t qui ont elles de meilleurs résultats que les actions de la politique courante u_t. Cette mise à jour repose sur l'erreur de différence temporelle $\delta_t = \big(r_{t+1} + \gamma V(s_{t+1})\big) - V(s_t)$. Étant donné que δ_t dépend de l'approximation de V, la mise à jour de l'acteur est accomplie avant celle du critique. Comme l'algorithme CACLA, la mise à jour est effectuée vers l'action exploratoire a_t seulement lorsque $\delta_t > 0$. Cependant, le processus n'est pas en ligne, mais utilise les données de $\mathcal{D}_\pi$. Une unique base de données $S \times A$ est suffisante puisqu'il n'y a pas de processus itératif comme avec le critique (pas d'auto-dépendance dans la formule) :

$$\hat{\pi}^* = \underset{\pi \in \mathcal{F}_a}{\operatorname{argmin}} \sum_{(s_t, u_t, a_t) \in \mathcal{D}_\pi} \begin{cases} \big(\pi(s_t) - a_t\big)^2, & \text{if } \delta_t > 0 \\ \big(\pi(s_t) - u_t\big)^2, & \text{otherwise} \end{cases} \tag{5}$$

Notez qu'un cas d'exploitation, où a_t est égale à u_t, pourrait également donner lieu à une mise à jour de la politique. Tandis que CACLA est *en ligne*, et ne peut que renforcer une action d'exploratoire, le cadre *hors ligne* et supervisé de NFAC permet de modifier plus profondément la politique en essayant de reproduire à l'exact cette action exploratoire.

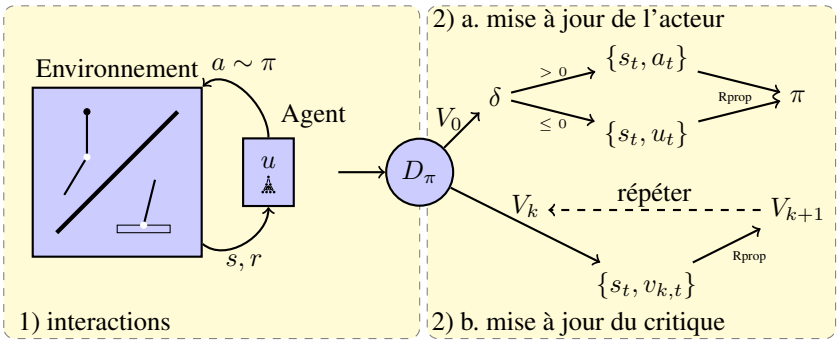

FIGURE 1 – Schéma de fonctionnement de l'algorithme NFAC.

4 Résultats expérimentaux

NFAC et CACLA sont comparés sur deux environnements continus et déterministes : Acrobot (Spong, 1995) et Cartpole (Riedmiller *et al.*, 2007). L'acrobot est un bras, de deux articulations, sous actionné. La première articulation ne peut pas exercer de couple contrairement à la seconde. Il commence toujours dans la même position (verticale basse). La fonction de récompense est définie comme étant 1) +1 is le but est atteint (position verticale haute), 2) la hauteur maximale normalisée de l'effecteur d'extrémité si 500 étapes sont atteintes, 3) 0 sinon.

Cartpole, ou le pendule inversé, est également constitué de deux articulations. La première articulation, auquel aucun couple ne peut être appliqué, est un point de pivot entre le chariot et le pendule. La seconde articulation contrôle le mouvement horizontal du chariot. La fonction de récompense est définie comme étant 1) -1 quand le pendule tombe au sol 2) +1 sinon.

Les simulations ont été réalisées avec l'Open Dynamics Engine (ODE) (Smith, 2005) et les paramètres suivants :

	Masse	Gravité	Moment	Durée étape
Unités	g	$m \cdot s^{-2}$	N	s
Acrobot	127, 127	9.81	$[-0.125; 0.125]$	0.01
Cartpole	85, 38	9.81	$[-5; 5]$	0.01

Comme l'illustre la figure 2, NFAC a de meilleurs performances que CACLA. Sur l'acrobot, 75% des agents NFAC atteignent leurs objectifs après seulement 151 épisodes là où les agents CACLA ont besoin de 544 épisodes. La médiane montre que la qualité des politiques trouvés par NFAC est meilleure. Après 1000 épisodes, 75 % des agents NFAC peuvent atteindre leur but après seulement 313 étapes contre 453 étapes pour CACLA.

Sur cartpole, 25% des agents NFAC peuvent préserver leur but pendant 500 étapes après 623 épisodes contre 1419 épisodes pour CACLA. La médiane de NFAC est meilleure que celle de CACLA pendant les premiers épisodes, puis converge après 2200 épisodes.

Les statistiques ont été réalisées sur 150 différents agents après une étape d'optimization des méta-paramètres pour chacun des algorithmes :

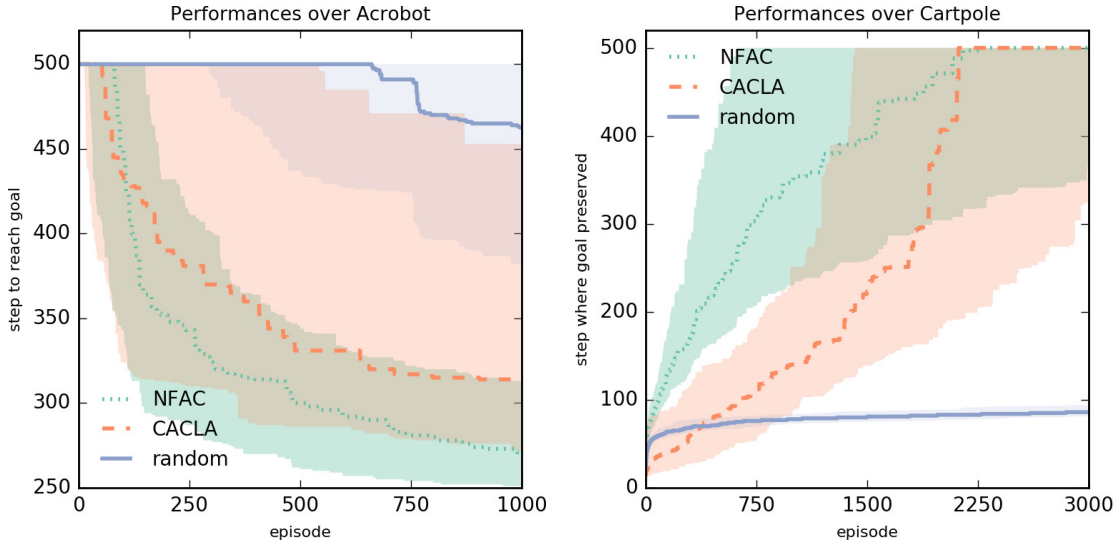

FIGURE 2 – Médiane et quartile de la meilleure performance enregistrée dans l'acrobot (meilleur en minimisant) et dans cartpole (meilleur en maximisant) durant l'apprentissage.

	CACLA Cartpole	NFAC Acrobot	CACLA Cartpole	NFAC Cartpole
Unités cachées acteur	5	5	5	5
Unités cachées critique	10	25	10	25
Taux apprentissage acteur	0.05		0.1	
Taux apprentissage critique	0.05		0.05	
iRprop- η^+		1.2		1.2
iRprop- η^-		0.5		0.5
iRprop- Δ_{max}		50		50
iRprop- Δ_{min}		0		0
Politiques ϵ-gloutonnes	0.05	0.05		
Politiques gaussiennes			0.4	0.4

Le nombre d'unités cachées pour le réseau de neurones de la politique, ainsi que la stratégie d'exploration est la même pour les deux algorithmes. Des politiques ϵ-gloutonne sont utilisés sur l'acrobot, et des politiques gaussiennes pour cartpole. Dans quelques environnements, CACLA s'est montré meilleur que CMA-ES et Natural Actor-Critic (Van Hasselt, 2012).

5 Conclusions et travaux futurs

NFAC est un algorithme acteur-critique, traitant des espaces continus, et utilisant des estimateurs non-linéaires à la fois pour le critique et l'acteur. Il s'inspire de CACLA et FQI. À la différence de FQI, NFAC peut être utilisé dans un environnement avec des actions continues. De plus, les expériences menées ont montré que NFAC fonctionnait mieux que CACLA avec moins de méta-paramètres. Pour améliorer encore NFAC, les données recueillies dans les épisodes précédents devraient être réutilisés. Pour le critique, cela signifie apprendre Q au lieu de V, la difficulté réside dans la façon d'utiliser les données précédentes pour la mise à jour de l'acteur. Enfin, cet algorithme pourrait être utilisé dans une perspective de robotique développementale puisqu'il ne nécessite que peu de connaissances prédéfinies.

Références

ASADA M., HOSODA K., KUNIYOSHI Y., ISHIGURO H., INUI T., YOSHIKAWA Y., OGINO M. & YOSHIDA C. (2009). Cognitive Developmental Robotics : A Survey. *IEEE Transactions on Autonomous Mental Development*, **1**(1), 1–44.

BRADTKE S. J., BARTO A. G. & KAELBLING P. (1996). Linear least-squares algorithms for temporal difference learning.

GRONDMAN I., BUŞONIU L., LOPES G. A. & BABUŠKA R. (2012). A survey of actor-critic reinforcement learning : Standard and natural policy gradients. *IEEE Transactions on Systems, Man and Cybernetics*, **42**(6), 1291–1307.

HANSEN N. & OSTERMEIER A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, **9**(2), 159–195.

IGEL C. & HÜSKEN M. (2000). Improving the Rprop learning algorithm. In *International Symposium on Neural Computation*, p. 115–121.

JOHN W. EATON DAVID BATEMAN S. H. & WEHBRING R. (2015). *{GNU Octave} version 4.0.0 manual : a high-level interactive language for numerical computations*.

KONDA V. R. & TSITSIKLIS J. N. (1999). Actor-Critic Algorithms. *Neural Information Processing Systems*, **13**, 1008–1014.

MNIH V., KAVUKCUOGLU K., SILVER D., RUSU A. A., VENESS J., BELLEMARE M. G., GRAVES A., RIEDMILLER M., FIDJELAND A. K. & OTHERS (2015). Human-level control through deep reinforcement learning. *Nature*, **518**, 529–533.

RIEDMILLER M. (2005). Neural fitted Q iteration - First experiences with a data efficient neural Reinforcement Learning method. In *Lecture Notes in Computer Science*, volume 3720 LNAI, p. 317–328.

RIEDMILLER M. & BRAUN H. (1992). RPROP - A Fast Adaptive Learning Algorithm. In *International Symposium on Computer and Information Science VII*.

RIEDMILLER M., PETERS J. & SCHAAL S. (2007). Evaluation of policy gradient methods and variants on the cart-pole benchmark. *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, p. 254–261.

SMITH R. (2005). Open dynamics engine.

SPONG M. W. (1995). Swing up control problem for the acrobot. *IEEE Control Systems Magazine*, **15**(1), 49–55.

SUTTON R. S. & BARTO A. G. (1998). *Reinforcement Learning : An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book.

SUTTON R. S., MCALLESTER D., SINGH S. & MANSOUR Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In Advances in Neural Information Processing Systems 12*, p. 1057–1063.

VAN HASSELT H. (2012). Reinforcement Learning in Continuous State and Action Spaces. In *Reinforcement Learning*, p. 207–251. Springer Berlin Heidelberg.

VAN HASSELT H. & WIERING M. A. (2007). Reinforcement learning in continuous action spaces. In *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, p. 272–279.